A joint communication and application simulator for NoC-based SoCs

†‡Carlo Condo, ‡Amer Baghdadi, †Guido Masera
† Politecnico di Torino, ‡ Telecom Bretagne

***Abstract*—NoCs have become a widespread paradigm in the system-on-chip design world, not only for multi-purpose SoCs, but also for application-specific ICs. The common approach in the NoC design world is to separate the design of the interconnection from the design of the processing elements: this is well suited for a large number of developments, but the need for joint application and NoC design is not uncommon, especially in the application specific case. The correlation between processing and communication tasks can be strong, and separate or trace-based simulations fall often short of the desired precision. In this work, the OMNET++ based JANoCS simulator is presented: concurrent simulation of processing and communication allow cycle-accurate evaluation of the system. Two cases of study are presented, showing both the need for joint simulations and the effectiveness of JANoCS.**

***Index Terms*—NoC, SoC, simulator, LDPC, turbo, genetic**

I. Introduction

In a general purpose System-On-Chip (SoC), one or multiple applications are mapped on a number of processing elements (PE), and communication among them is handled by interconnects. Advanced interconnects use the Networks-on-Chip (NoC) concept, which potentially guarantees very high flexibility and better scalability with respect to traditional bus based solutions. In the NoC design paradigm, processing is separated from communication, meaning that supported applications can be developed and optimized independently of the underlying interconnect structure. On the other hand, NoC design choices are made with the purpose of matching the communication requirements (e.g. latency, aggregate throughput, FIFO sizes, etc.) of a large variety of applications. While this separation is still important to facilitate development of new applications and reduce the time to market of new products, the original idea of general purpose NoCs has evolved and new kinds of NoCs are today proposed with features fully optimized for a single or reduced number of processing needs (Application Specific NoCs, or ASNoC). In this context, the separation paradigm looses part of its meaning and proper design methods and tools, involving both communication and processing sides must be proposed to support the development of efficient ASNoCs.

Many NoC simulators have been proposed and a great majority of them is intended for general purpose NoCs. For example, in [1] and [2] the Nostrum simulator is used for NoC design and optimization respectively. Test applications, if taken in consideration, are usually run only as a verification step. In most cases, random generators or application traces are used for traffic modeling: in [3] message dependencies are introduced in the traces for more reliable simulation, a technique subsequently perfected in [4]. Very few simulators offer the ability to jointly model both application and communication across the network: the NOCSEP platform [5] is aimed at early design space exploration and provides fine-grained application modeling. In the NOCSEP platform, cycle approximate simulation runs are supported by means of traffic patterns that take into account the latency introduced by the PEs. However, in this approach, some features of the PEs are captured in the simulation environment with the purpose of optimizing the performance of the NoC and a true joint simulation of processing and communication is not supported.

In this paper, we show that some cases arise where cycle accurate joint simulation of PEs and NoC is necessary to achieve efficient application specific, NoC-based VLSI architectures. We distinguish three kinds of developments, to clarify the possible use of such a joint simulation environment:

1) new NoC designed for an already existing set of PEs associated with a given application
2) new application with new PEs to be interconnected by means of an already available NoC
3) new NoC and a new application, supported by new PEs.

In the first case, the inter PE communication needs are extracted from the application and used to drive the NoC design. Available simulation tools already support this kind of design flow, where effects of application on NoC performance must be verified. We can symbolically represent this dependency as: PEs $\rightarrow$ NoC. As an example, in [6] a complex digital TV SoC is extended by replacing the original interconnect structure with a programmable NoC, which provides improved flexibility and extends the lifecycle of the product. On the contrary, in case two, choices on the processing side depend on the capabilities of the available NoC: in this case we have a NoC $\rightarrow$ PEs dependency. Finally, the third case refers to the design of a new ASNoC, which is usually addressed exploiting the separation principle: first PEs are developed able to run in a distributed way the application, then a proper NoC is derived, starting from the specific communication needs coming from the PEs. In other words, the case three development is usually organized around a PEs $\rightarrow$ NoC dependency. We believe that, in some cases, better results can be achieved if PEs and NoC are jointly designed, taking in account both the effects of generated inter PE traffic on NoC performance, and the effects of NoC design choices on the overall application performance. We can represent this inter dependency as PEs $\leftrightarrow$ NoC. A

key tool to support joint PE and NoC design is a high level simulation environment including proper modeling of both PEs and NoC.

In this paper, starting from the OMNET++ library [7], a new simulation environment is proposed, which allows for cycle-accurate joint simulations of both application and communication in NoC-based SoCs. Simulations are run at high level and do not require RTL modeling. Its usage is fit for early evaluation of design choices for all kinds of NoC-based SoCs, complete design of multiprocessor ASICs, and evaluation of feasibility in case a new application is mapped on existing hardware. Two design cases are presented to show the advantages of the joint simulation.

II. Proposed simulation environment

The idea behind the proposed simulator is that the simulation of PEs and interconnects must be handled at the same time, allowing to observe the impact of the network dynamic behavior on the application mapped over the PEs. The range of possible applications is wide and diversified, but common needs allow to draw a list of characteristics that the simulator should have:

- a high-level language must be used to simplify the modeling of the system and to improve design productivity;
- the overall structure should be as modular as possible, allowing fast and easy changes to the model, and guaranteeing a high degree of reusability and flexibility;
- hardware characteristics should be modeled with functional behavior and timing details only, to keep simulations fast without sacrificing precision.

An effective starting point for these purposes is the OMNET++ simulation library [7], an open-source C++ based environment targeting network simulation. It is composed of two different parts:

1) the core is a set of C++ classes representing network elements (including both computational hardware and data structures) and useful methods, which can be used to model the behavior of the network itself;
2) the structure is described with a NEtwork Description (NED) language; here modules are interconnected, specifying channel types, data rates, topologies.

Parameters used in both parts are defined by the user through an initialization file.

The key feature of OMNET++ lies in its information exchange method: with the help of the `cMessage` class, behavioral responses of the modules are triggered at the reception of different kinds of messages. Messages are moved between modules via the channels defined with NED: through a set of customizable parameters, statistical data can be easily obtained and plotted. Correct modeling of the channels' timing details, together with the inset message-triggered responses, allow cycle-accurate behavioral simulations and a complete control over the status of the network. Latencies, congestion, data loss and many other effects and characteristics can be easily evaluated.

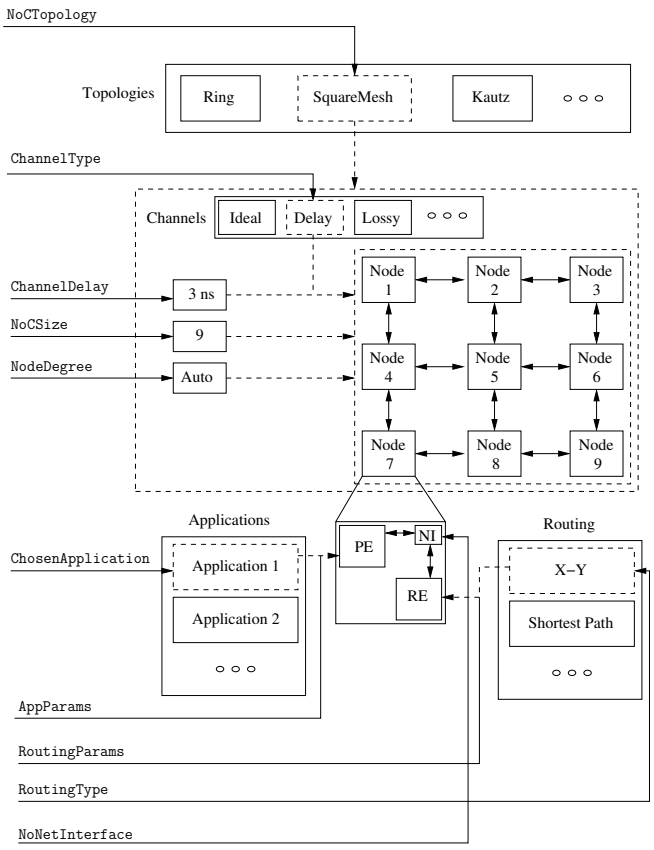

Figure 1. JANoCS network definition via parameters

With OMNET++ as a foundation, a Joint Application and Network-on-Chip Simulator (JANoCS) is proposed, targeting the above listed characteristics. The NED structure of the NoC has been developed to be completely parameterized: topology, number of nodes, direction of the channels and their delays are passed as parameters via the initialization file. Figure 1 shows an example of JANoCS network definition via parameters. The first parameter passed is the topology of the NoC: a different construction file is present for every supported topology, defining the connection among nodes accordingly. Once the topology has been specified, a set of variables common to all topologies are given to the construction file, like the number of nodes, the number of connections (degree) and the type of channels to be used for each connection (delayed, lossy, ideal, etc.).

The construction file instantiates the nodes of the NoC as generic JANoCS blocks, *i.e.* compound modules made of a Routing Element (RE), a PE and a network interface (NI). This compounds are provided with another set of user-defined parameters, specifying which modules have to be used. In Fig. 1, the chosen router implements XY routing: a further set of parameters clarifies characteristics that can either be common to all routers or particular to the selected module, like priority-based collision management or definition of the

size of the input FIFOs. The same can be said for the NI block. The PE module defines the application that is going to be simulated: each possible PE requires a dedicated set of parameters, according to the type of processing performed.

Exploiting the message-triggered responses intrinsic to OMNET++ models, a clock generator module has been created, with user-defined period. Each module instantiated receives one or more latency parameters, expressed in number of clock cycles: this allows for specific delays associated to packets travelling through the NoC and to input/output connections. These latencies are used to simulate the hardware structure within the models, with no need for a more detailed datapath description.

This extremely modular structure of the system makes very simple the introduction of new modules and their customization. Each block is instantiated by a compound module as a generic class, and consequently identified by the user as a subclass of the instance: for example, both `XYRouter` and `ShortPathRouter` classes are subclasses of `RoutingElement`, thus inheriting the parameters common to all REs, and specifying new ones. Moreover, changes within the low-level modules are often reduced to differences in function calls: they can either be implemented as parameters within the same multi-purpose module, or as different module constructors to keep functionalities separated.

The potential of JANoCS can be better explained through an example. For this purpose, an optimization task based on a genetic algorithm has been mapped on a NoC with `NoCSize`=16. This example situation, named *barrel construction*, is very simple, but does not alter the generality of the concept.

The goal of the optimization task is building the largest barrel possible, using a fixed number of planks and wasting as little material as possible: having very long planks when a short plank is present is considered a waste of resources. In genetic algorithms the fitting function expresses how good each member of the current population is: in this particular problem it has been defined as

$$f_{fit} = \frac{\sum_{i=0}^{p} l_i}{l_{max} - l_{min}} \qquad (1)$$

where l_i is the length of the i^{th} plank in the barrel, the numerator is the whole size, while the denominator expresses the wasted wood.

Each PE is modeled to hold a small randomly generated initial population of `PopulationSize` barrels: new barrels are created through crossover and mutation (with probabilities `CrossoverProbability` and `MutationProbability`). Then the PEs broadcast a few of the best barrels to all the others PEs, speeding up the improvement process. The PEs are coded to wait `StallingTime` cycles after the last barrel has been sent, to allow the delivery of the barrels: when they have expired, another iteration (crossover - mutation - broadcast) starts. Iterations are stopped when a solution with fitting value higher

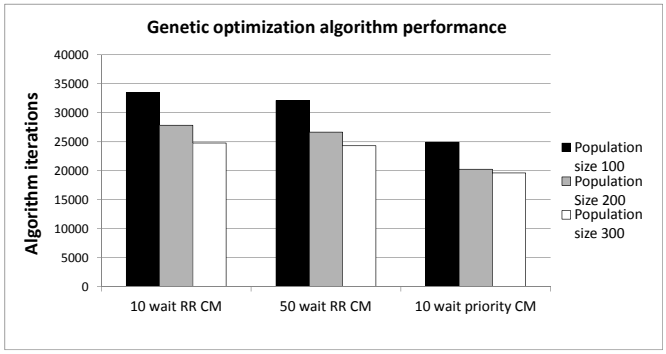

Figure 2. Performance of genetic algorithm based barrel construction

than `FittingThreshold` is found: for this example, `FittingThreshold`=1000. A manager module can conveniently control the status of the application and order the start of iterations: dedicated `cMessage` subclasses can be used as control messages.

Communication is enforced via a 4×4 square mesh NoC with XY routing: `NodeDegree` is automatically set to 5, while `ChannelType` is set to Ideal. Routers can easily be built from basic `cQueue` blocks representing FIFOs, with functions implementing the actual routing decisions, timed by latency parameters. Change of routing algorithm or arbitration method implies minimal changes to the code.

Joint simulations allow to evaluate the importance of inter-PE communication and possibly to improve the design towards a faster convergence to a good solution. In Figure 2 the number of algorithm iterations required to reach the fitting threshold is shown for different design choices. The first three histogram bars refer to a NoC implementing a round-robin collision management policy (RRCM), barrels built out of 12 planks, `StallingTime`=10 clock cycles, `CrossoverProbability`=30%, and `MutationProbability`=0.5%. The black bars refer to a population size of 100 elements, while the gray and white bars are for 200 and 300 elements respectively. Larger populations usually lead to larger room for improvement through crossover and mutation, thus explaining the faster convergence towards the desired fitting threshold: on the downside, they require additional PE resources and longer iterations. The second block of solutions considers an extended `StallingTime` of 50 cycles: it can be seen that although small improvements can be noticed, the congestion of the network is too severe, and would require a much longer time to be drained. In the third block, a 10-cycle `StallingTime` is assumed again, but priority is introduced in the handling of packets (priority CM), particularly in the management of collisions at the routers. The convergence is faster of averagely a 25% factor thanks to the higher number of best barrels reaching destination, and improving the population of each PE.

These results are meaningful for both the interconnection and the processing parts of the MP-SoC: the interdependence between the application and the network is evident, and joint

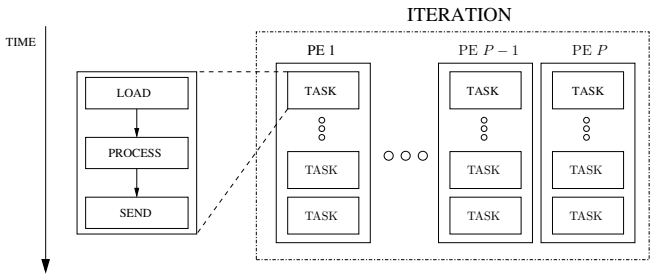

Figure 3. LDPC and turbo codes distributed decoding pattern

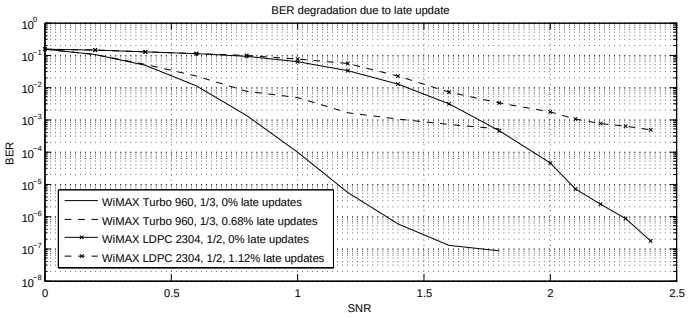

Figure 4. BER degradation due to late information update

simulation is necessary for both validating the design and reducing its overhead.

III. LDPC AND TURBO DECODING: A CASE OF STUDY

LDPC [8] and turbo [9] codes are forward error-correcting codes whose study, usage and implementation have seen a steep increase in the last decade. Standards supporting broad sets of one or both code types have risen and evolved, creating a large pool of requirements. In particular the decoding process, being based on computationally intensive iterative algorithms, poses severe requirements on both computational power and latency, making the hardware design a challenging task.

LDPC and turbo codes decoding rely on iterative algorithms: during each iteration, information is processed, updated and exchanged. Message passing is a key component of the decoding process, especially in parallel implementations: the algorithm is in fact mapped on different PEs. A critical role is consequently played by the structure interconnecting the various processing elements: low-complexity and low-latency networks must be devised, with the necessary degree of flexibility to support the decoding of multiple codes, code types, and even standards. There are many possible ways to tackle the interconnection issues in both codes, but the NoC approach has been considered and proven effective (e.g. [10], [11]).

Although different in nature, both LDPC and Turbo decoding, particularly when considering distribute decoding, share a common pattern, summarized in Fig. 3. A set of data is loaded from the memory of each PE, processed according to the decoding algorithm, and sent through the interconnection structure to another PE based on communication needs associated to each specific code. While the information is traveling, another set of data is loaded from the PE memories, starting the process again: this flow is repeated for all the scheduled tasks until an iteration is completed, and several iterations might be necessary for a correct decoding. Iterations in interconnect structures traditionally used to support channel decoding (e.g Butterfly or Benes networks) end when the last message has reached its destination, and the introduced delay is deterministic. In the NoC approach, however, the delay can change with every message: this affects the decoder's performance in terms of both throughput and bit error rate (BER). The message delivery time is not deterministic and therefore the overall decoder throughput depends on the network status, which changes dynamically during the processing. In order to avoid excessive delivery times, messages can be discarded when the accumulated delay across the NoC passes a given threshold. This implies that the destination PE must execute its decoding task without the discarded message: in this case, either the previous version of the discarded message, the one that was received at the previous iteration, can be used, or a predefined constant is assumed as a replacement. Both solutions deeply affects BER performance: it can be seen in Figure 4 that a small percentage of discarded messages (late updates) causes a huge performance loss. The relevance of this effect also changes with the considered code and with the implemented decoding algorithm.

Possible solutions to the problem include priority-based routing and inevitably intertwine further processing and communication. Thus, in order to carefully evaluate the effects on achievable performance of each design choice in both NoC and PEs, a joint application and communication simulation is necessary. For this purpose, LDPC and turbo PE modules have been modeled, together with a set of unique modules handling initialization, iteration count and result gathering.

A. PE modules

The LDPC PE is composed of a processing module and two memory modules, one used to store the incoming messages (data memory), while the other is initialized at startup with the destination of the outgoing messages (interleaving memory). Since the communication graph is known and static once the topology and the code have been selected, the interleaving memory is written only once per simulation. The memories are parametrized blocks designed to load and store `cMessage` objects and their subclasses: for the LDPC decoding modules, the class expressing the packets traveling on the NoC is `LDPCMessage`. The width and depth of the memories is decided by the user at initialization time, allowing a single model to be used in very different implementations: the need for remodeling in case new fields are added to the `LDPCMessage` class is consequently eliminated, and the same models can also be used in the turbo codes PE. The processing module of the LDPC PE simulates the implementation of a serial pipelined structure similar to the architecture proposed in [10]. Two phases occur concurrently: data needed for the processing

of a task are loaded from the data memory, while messages computed within the former task are injected in the NoC. The actual computation itself is instantaneous, occurring just after the last message has been retrieved from the memory: latency parameters are used to model the memory access and the processing hardware. In particular, two sets of three latencies are used to build the outgoing message's queue and the memory access requests (initial, inter-message and final latencies).

The turbo codes decoding module (known as SISO, or Soft-Input Soft-Output, unit) is characterized by two operating modes, the selection of which is handled at initialization time via parameter. Each of them supposes different hardware resources, resulting in diverse packet injection rates and achievable throughputs. The SISO module makes use of three memory modules: an interleaving memory and a data memory, analogous to those already described in the LDPC PE, and a separate intrinsic memory to store channel intrinsic information. This extra module is necessary since, unlike LDPC decoding, extrinsic are not allowed to overwrite channel intrinsic metrics, that are necessary throughout the whole decoding process.

Three additional modules are necessary for both decoding tasks:

- *Initializer*: this block handles the channel simulation: source bits are created, encoded and sent through a channel according to the selected signal-to-noise ratio (SNR) points. The resulting Logarithmic Likelihood Ratios (LLRs), *i.e.* measures of the error probability on each bit, initialize the PE data and intrinsic memories. This process must be repeated for all the SNR points and all the frames considered by a simulation. Moreover, once per simulation the module handles the mapping of the tasks over the topology and the consequent initialization of the interleaving memories.
- *Global decider*: this module is activated at the end of every iteration. It gathers the latest information about each bit of the frame in the local PE memories, on which the decision on the bits is based. The resulting received frame is consequently compared with the sent frame, provided by the initializer: errors are counted and notified to the decoding manager.
- *Decoding manager*: it monitors the status of both network and PEs to detect the end of an iteration, triggering the global decider. The BER is updated in case of correct decoding or reached iteration limit.

B. Custom NoC

The NoC modeled for this particular case of study is inspired to the ASNoC implemented in [10] for a flexible turbo/LDPC code decode. REs are constituted of an $F \times F$ crossbar switch, with F input FIFOs and F output ports: control logic or routing tables implement the routing algorithm, and no additional network interface is present, since packet construction is handled by the PE.

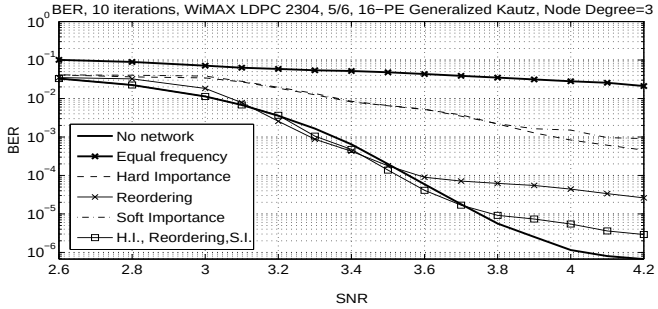

Figure 5. Simulated BER curves - LDPC code example

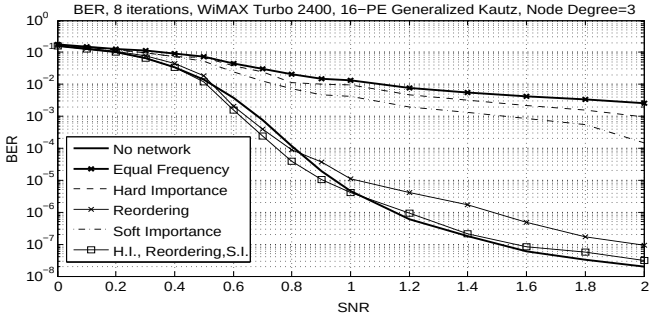

Figure 6. Simulated BER curves - turbo code example

The REs modeled for this particular case of study consider the shortest paths among the nodes, internally storing the information during the initialization phase of the simulation. As with the PE, a set of latencies is passed as parameters by the user to provide correct timing information.

IV. Simulations

The simulation space that LDPC and turbo code decoding provide is very wide: changes in code length, code rate, NoC size, RE degree, latencies and clock frequency all account for diverse communication performance. A few early choices based on previous works have consequently been made.

The selected NoC topology is the generalized Kautz network [12] with router degree 3, whose qualities have already been pointed out in [10], [13]: in these papers, a wide set of topologies is compared in terms of interconnect efficiency and implementation area, with the Kautz emerging as the best trade-off between speed and area. A first batch of simulations have been run, some of the results already addressed in Figure 4. From these preliminary simulations, the strong connection between code size and number of PEs has been made evident: a small code mapped on a large NoC suffers the worst BER degradation. Indeed, the time available for packet delivery before an information is needed is proportional to the code size: since large NoCs have longer diameters, the number of late packets tends to increase. The plots presented in Figures 5 and 6 consider large codes mapped on medium-size NoCs, where the dominating factor in late update is network congestion, and not node distance.

The high throughput requirements usually imposed by standards employing LDPC and turbo codes imply a high

message injection rate in the NoC. Consequently, stalling the decoding waiting for late packets is not acceptable. To tackle the late update problem from another angle, two substantial modifications have been applied to the model. The first one is the urgency-based packet priority: each PE is able to perform an estimate, based on local information, of the time available for delivery of each outgoing packet (precise in turbo codes, more approximate in LDPC codes). A measure of this time is attached to each packet header, and used in collision management by the routers: priority is given to the most urgent packet, while the urgency is updated at every clock cycle. Packets that reach zero available time are automatically discarded, avoiding unnecessary network traffic. All curves plotted in Fig. 5 and 6 implement the urgency-based packet priority.

Another additional feature is the separation of clocks and scaling of frequencies. In [10] the possibility of different NoC and PE frequencies was partially explored, since an higher NoC frequency would be equivalent to a reduced packet injection rate, and would consequently smooth the network traffic. Two clock source modules feed separately PEs and REs (f_{PE} and f_{NoC}), their frequencies set by the user, with the possibility of lowering the ratio f_{NoC}/f_{PE} after a certain amount of iterations to reduce power consumption. In Fig. 5, apart from a reference theoretical BER curve ("no network") and the "equal frequency" plot, the other curves have been obtained by changing the f_{NoC}/f_{PE} ratio from two to one after two iterations have been completed. The same can be said for Fig. 6, with f_{NoC}/f_{PE} changed from 10/8 to one.

With these premises, a set of traffic reduction methods have been explored.

- *Hard and Soft Importance*: with the hard importance (dashed line in Fig. 5 and 6) each PE evaluates the content of the outgoing packets, assessing the valuableness of the carried information. If it is not essential (unchanged from previous iteration, saturated value, etc.) the packets are not sent. Soft importance (dash-dot line) implements the same kind of concept, with more relaxed thresholds: packets are sent anyway, but flagged as expendable. In case of collisions, the expendable packets can be discarded by the routers. These methods alone do not manage to bring the BER plot close to the theoretical limit, still accounting for large degradation: however they contribute to a large reduction in traffic, with an average of 30% less exchanged messages each iteration in both turbo and LDPC decoding.
- *FIFO reordering*: with this method packets are sorted in order of urgency on arrival to a router's FIFO. In this way, very urgent packets have greater chances of being delivered on time, since they are always the first ones to be popped from the FIFO. In both Fig. 5 and 6 the cross-marked line represents FIFO reordering alone, and while being quite close to the reference plot, some degradation is still present, especially in the LDPC case. When working together with hard and soft importance (square-marked line), the degradation is negligible, particularly in turbo codes.

From the results obtained in this complex design case, it can be said that JANoCS allows to correctly assess the impact of NoC communication on the addressed application, to unveil critical issues and to evaluate the advantages provided by different solutions in both NoC and PE. As the proposed environment operates on high level cycle accurate models, it enables a fast exploration of several alternatives.

V. Conclusions

The JANoCS simulator has been presented, allowing easy and reusable modeling of NoC-based SoCs and joint simulation of both application and communication. The developed tool has been used to simulate the LDPC and turbo parallel decoding applications, revealing a particularly critical problem, and exploring a set of possible solutions: results obtained show BER curves very close to the theoretical limit and significantly reduced NoC traffic. Future works foresee both hardware synthesis of the simulated models and extended exploration of the design space. The proposed simulation tool has been also applied to a smaller example, dealing with genetic algorithms, and can be used in similar design cases, where processing and communication need to be jointly optimized.

References

[1] M. Millberg, E. Nilsson, R. Thid, S. Kumar, and A. Jantsch, "The Nostrum backbone-a communication protocol stack for Networks on Chip," in *VLSI Design, 2004. Proceedings. 17th International Conference on*, 2004, pp. 693 – 696.

[2] L. Papadopoulos, S. Mamagkakis, F. Catthoor, and D. Soudris, "Application - specific NoC platform design based on System Level Optimization," in *VLSI, 2007. ISVLSI '07. IEEE Computer Society Annual Symposium on*, march 2007, pp. 311 –316.

[3] F. Trivino, F. Andujar, F. Alfaro, J. Sanchez, and A. Ros, "Self-related traces: An alternative to full-system simulation for NoCs," in *High Performance Computing and Simulation (HPCS), 2011 International Conference on*, july 2011, pp. 819 –824.

[4] Y.-C. Huang, Y.-C. Chang, T.-C. Tsai, Y.-Y. Chang, and C.-T. King, "Attackboard: A novel dependency-aware traffic generator for exploring NoC design space," in *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE*, june 2012, pp. 376 –381.

[5] C.-F. Chang and Y. Hsu, "A system exploration platform for Network-on-Chip," in *Parallel and Distributed Processing with Applications (ISPA), 2010 International Symposium on*, sept. 2010, pp. 359 –366.

[6] F. Steenhof, H. Duque, B. Nilsson, K. Goossens, and R. Llopis, "Networks on chips for high-end consumer-electronics TV system architectures," in *Design, Automation and Test in Europe, 2006. DATE '06. Proceedings*, vol. 2, march 2006, pp. 1 –6.

[7] OMNET++ network simulation framework. [Online]. Available: http://www.omnetpp.org/

[8] R. G. Gallager, "Low density parity check codes," *IRE Transactions on Information Theory*, vol. IT-8, no. 1, pp. 21–28, Jan 1962.

[9] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error correcting coding and decoding: Turbo codes," in *IEEE International Conference on Comm.*, 1993, pp. 1064–1070.

[10] C. Condo, M. Martina, and G. Masera, "A network-on-chip-based turbo/LDPC decoder architecture," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, march 2012, pp. 1525 –1530.

[11] H. Moussa, A. Baghdadi, and M. Jezequel, "Binary De Bruijn onchip network for a flexible multiprocessor LDPC decoder," in *ACM/IEEE Design Automation Conference*, 2008, pp. 429–434.

[12] M. Imase, M.; Itoh, "A design for directed graphs with minimum diameter," *Computers, IEEE Transactions on*, vol. C-32, no. 8, pp. 782–784, Aug. 1983.

[13] M. Martina and G. Masera, "Turbo NOC: A framework for the design of network–on–chip–based turbo decoder architectures," *IEEE Trans. on Circuits and Sistems I*, vol. 57, no. 10, pp. 2776 – 2789, 2010.